

K-Regret Queries: From Additive to Multiplicative Utilities

Jianzhong Qi¹, Fei Zuo^{1*}, Hanan Samet², Jia Cheng Yao³

¹Department of Computing and Information Systems, University of Melbourne

²Department of Computer Science, University of Maryland

³Faculty of Business and Economics, University of Melbourne

jianzhong.qi@unimelb.edu.au, fzuozuo@student.unimelb.edu.au, hjs@cs.umd.edu, yaoj1@student.unimelb.edu.au

ABSTRACT

The k -regret query aims to return a size- k subset of the entire database such that, for any query user that selects a data object in this size- k subset rather than in the entire database, her *regret ratio* is minimized. Here, the regret ratio is modeled by the level of difference in the *optimality* between the optimal object in the size- k subset returned and the optimal object in the entire database. The optimality of a data object in turn is usually modeled by a *utility function* of the query user. Compared with traditional top- k queries, k -regret queries have the advantage of not requiring users to specify their utility functions. They can discover a size- k subset that minimizes the regret ratio for a whole family of utility functions without knowing any particular of them. Previous studies have answered k -regret queries with additive utility functions such as the linear summation function. However, no existing result has been reported to answer k -regret queries with multiplicative utility functions, which are an important family of utility functions.

In this study, we break the barrier of multiplicative utility functions. We present an algorithm that can produce answers with a bounded regret ratio to k -regret queries with multiplicative utility functions. As a case study we apply this algorithm to process a special type of multiplicative utility functions, the Cobb-Douglas function, and a closely related function, the Constant Elasticity of Substitution function. We perform extensive experiments on the proposed algorithm. The results confirm that the proposed algorithm can answer k -regret queries with multiplicative utility functions efficiently with a constantly small regret ratio.

1. INTRODUCTION

Top- k queries [7, 8, 19] and *skyline queries* [1, 15, 20] have been used traditionally to return a representative subset \mathcal{S} of a database \mathcal{D} to a query user when the entire database is too large to be explored fully by the query user. These two

*Co-first author ordered by surname.

Table 1: A Computer Database

Computer	CPU ($p_i.c_1$)	Brand recognition ($p_i.c_2$)
p_1	2.3	80
p_2	1.7	90
p_3	2.8	50
p_4	2.1	55
p_5	2.1	50
p_6	3.0	55

types of queries suffer in either requiring a predefined *utility function* to model the query user's preference over the data objects, or returning an unbounded number of data objects. Recent studies [10, 14, 26] aim to overcome these limitations by a new type of queries, the k -regret query, which returns a size- k subset $\mathcal{S} \subseteq \mathcal{D}$ that minimizes the *maximum regret ratio* of any query user. The concept of *regret* comes from microeconomics [11]. Intuitively, if a query user had selected the local optimal object in \mathcal{S} , and were later shown the overall optimal object in the entire database \mathcal{D} , the query user may regret. The k -regret query uses the *regret ratio* to model how regretful the query user may be, which is the level of difference in the *optimality* between the local optimal object in \mathcal{S} and the overall optimal object in \mathcal{D} . Here, the optimality is computed by a utility function. The k -regret query does not require any specific utility function to be given. Instead, it can return a set to minimize the maximum regret ratio for a family of utility functions such as the linear summation functions.

To illustrate the k -regret query, consider an online computer shop with a database \mathcal{D} of computers as shown in Table 1. There are six computers, i.e., $\mathcal{D} = \{p_1, p_2, \dots, p_6\}$. Every computer p_i has two attributes: CPU clock speed and brand recognition, denoted as $p_i.c_1$ and $p_i.c_2$, respectively. Here, brand recognition represents how well a brand is recognized by the customers. A larger value means that the brand is better recognized. Since the entire database may be too large to be all shown, the shop considers showing only a size- k subset $\mathcal{S} \subseteq \mathcal{D}$ in the front page as a recommendation. Such a subset may be $\mathcal{S} = \{p_1, p_3, p_5\}$ (i.e., $k = 3$). When a customer visits the shop, assume that her preference can be expressed as a utility function $f_1(p_i) = 0.5 \cdot p_i.c_1 + 0.5 \cdot p_i.c_2$. Then the customer may purchase p_1 from the recommended subset \mathcal{S} since p_1 has the largest utility value $f_1(p_1) = 0.5 \cdot p_1.c_1 + 0.5 \cdot p_1.c_2 = 0.5 \times 2.3 + 0.5 \times 80 = 41.15 > f_1(p_3) = 26.40 > f_1(p_5) = 26.05$. Note that another computer $p_2 \in \mathcal{D}$ exists with an even larger utility value $f_1(p_2) = 0.5 \times 1.7 + 0.5 \times 90 = 45.85$. If the customer later sees p_2 , she may regret. Her regret ratio is computed

as $\frac{f_1(p_2) - f_1(p_1)}{f_1(p_2)} \approx 10.25\%$. For another customer with a different utility function $f_2(p_i) = 0.99 \cdot p_i.c_1 + 0.01 \cdot p_i.c_2$, the computer in \mathcal{S} that best suits her preference is p_3 , i.e., $f_2(p_3) = 0.99 \times 2.8 + 0.01 \times 50 \approx 3.27 > f_2(p_1) \approx 3.08 > f_2(p_5) \approx 2.58$. Meanwhile, the overall best computer in \mathcal{D} is p_6 , i.e., $f_2(p_6) = 0.99 \times 3.0 + 0.01 \times 55 = 3.52$. If the customer purchases p_3 , her regret ratio will be $\frac{f_2(p_6) - f_2(p_3)}{f_2(p_6)} \approx$

7.05%. Since every customer may have a different preference and hence a different utility function, it is impractical to satisfy multiple customers with a single set \mathcal{S} . Instead of trying to satisfy some customer with a certain utility function, the k -regret query aims to generate a subset \mathcal{S} that minimizes the *maximum regret ratio* for a family of (infinite) utility functions.

Existing studies on the k -regret query have focused on *additive utility functions* where the overall utility of an object is computed as the sum of the utility in each attribute of the object. The linear summation functions f_1 and f_2 are examples. They can be written in a more general form: $f(p_i) = \sum_{j=1}^d \alpha_j \cdot p_i.c_j$, where d denotes the number of attributes, and α_j is the *weight* of attribute j . Studies [10, 14] have shown that the maximum regret ratio of the k -regret query with additive utility functions can be bounded. However, to the best of our knowledge, so far, no existing bound has been obtained for the k -regret query with *multiplicative utility functions* (MUFs).

In this paper, we break the barrier of bounding the maximum regret ratio for the k -regret query with MUFs. An MUF computes the overall utility of an object as the product of the utility in each attribute, i.e., $f(p_i) = \prod_{j=1}^d p_i.c_j^{\alpha_j}$. It is more expressive especially in modeling the *diminishing marginal rate of substitution* (DMRS) [22]. The DMRS is a common phenomenon. It refers to the fact that, as the utility in an attribute j gets larger, the extent to which this utility can make up (substitute) for the utility in any other attribute j' decreases (diminishes). For example, a higher CPU clock speed may make up for a less known brand. However, as the CPU clock speed gets higher, the extent to which it can make up for the brand recognition decreases, since most customers only need a moderate CPU rather than an extremely fast one. This phenomenon cannot be modeled by an additive utility function such as f_1 or f_2 , where a given amount of increase in attribute 1 always makes up for a fixed amount of utility in attribute 2, and vice versa. As a result, f_1 and f_2 favor objects with maximum values in certain attributes (e.g., p_2 in c_2 and p_6 in c_1). An MUF such as $f_3(p_i) = p_i.c_1^{0.5} \cdot p_i.c_2^{0.5}$, on the other hand, models the DMRS much better. It favors objects with a good (but not necessarily the best) utility in more attributes, e.g., p_1 suits f_3 the best, which is reasonably good in both attributes.

The higher expressive power of MUFs also brings significant challenges in bounding the maximum regret ratio for them. It is much more difficult to tightly bound the product of a series of exponential expressions. In this study, we overcome these challenges with an algorithm named *MinVar*. We show that this algorithm can obtain a maximum regret ratio bounded between $\Omega(\frac{1}{k^2})$ and $O(\ln(1 + \frac{1}{k^{\frac{1}{d-1}}}))$ for the k -regret query with MUFs. To showcase the applicability of the MinVar algorithm in real world scenarios, we apply it on k -regret queries with a special family of MUFs,

the *Cobb-Douglas functions*, which are used extensively in economics studies [4, 6, 24]. As a by-product, we obtain a new upper bound $O(\frac{1}{k^{\frac{1}{d-1}}})$ of the maximum regret ratio for the k -regret query with the *Constant Elasticity of Substitution* (CES) functions [24]. This type of function is closely related to the Cobb-Douglas functions. Our upper bound is tighter than a previously obtained upper bound [10].

To summarize, this paper makes the follows contributions:

- We are the first to study the k -regret query with multiplicative utility functions.
- We propose an algorithm named MinVar to process the query. Based on this algorithm, we obtain bounds of the maximum regret ratio for the k -regret query with multiplicative utility functions. We introduce an extra heuristic based on redundancy elimination to further lower the maximum regret ratio, which results in an improved algorithm named RF-MinVar.
- We showcase the applicability of the proposed algorithms on a special type of multiplicative utility functions, the Cobb-Douglas functions, and a closely related type of functions, the CES functions.
- We perform extensive experiments using both real and synthetic data to verify the effectiveness and efficiency of the proposed algorithms. The results show that the regret ratios obtained by the proposed algorithms are constantly small. Meanwhile, the proposed algorithms are more efficient than the baseline algorithms.

The rest of the paper is organized as follows. Section 2 reviews related studies. Section 3 presents the basic concepts. Section 4 describes the proposed algorithms to process k -regret queries with MUFs, and derives the bounds of the maximum regret ratio. Section 5 applies the proposed algorithms on two special types of utility functions, the Cobb-Douglas function and the CES function, and shows their bounds of the maximum regret ratio. Section 6 presents experimental results and Section 7 concludes the paper.

2. RELATED WORK

We review two groups of related studies: the skyline queries and the k -regret queries.

Skyline queries. The skyline query [1] is an earlier attempt to generate a representative subset \mathcal{S} of the entire database \mathcal{D} without the need of specifying any utility functions. This query assumes a database \mathcal{D} of d -dimensional points ($d \in \mathbb{N}_+$). Let p_i and p_j be two points in \mathcal{D} where $i \neq j$. Point p_i is said to *dominate* point p_j if and only if $\forall l \in [1..d], p_i.c_l \geq p_j.c_l$, where $p_i.c_l$ ($p_j.c_l$) denotes the coordinate of p_i (p_j) in dimension l . Here, the “ \geq ” operator represents the user preference relationship. A point with a larger coordinate in dimension l is deemed more preferable in that dimension. The skyline query returns the subset $\mathcal{S} \subseteq \mathcal{D}$ where each point is *not* dominated by any other point in \mathcal{D} .

The skyline query can be answered by a two-layer nested loop over the points in \mathcal{D} and filtering those dominated by other points. The points remaining after the filtering are the *skyline points* which are the query answer. More efficient algorithms have been proposed in literature [15, 20].

While the skyline query does not require a utility function, it suffers in having no control over the size of the set \mathcal{S} returned. In the worst case, the entire database \mathcal{D} may be returned. Studies have tried to overcome this limitation by combining the skyline query with the top- k query. For example, Xia et al. [25] introduce the ε -skyline query which adds a *weight* to each dimension of the data points to reflect users' preference towards the dimension. The weights create a built-in rank for the points which can be used to answer the *top- k skyline query*. Chan et al. [3] rank the points by the *skyline frequency*, i.e., how frequently a point appears as a skyline point when different numbers of dimensions are considered. A few other studies extract a representative subset of the skyline points. Lin et al. [12] propose to return the k points that together dominate the largest number of non-skyline points as the *k most representative skyline subset*. Tao et al. [21] select *k representative skyline points* based on the distance between the skyline points instead. While the studies above return a size- k subset of the entire database, the maximum regret ratio of such a subset is unbounded.

K -regret queries. Nanongkai et al. [14] introduce the concept of *regret minimization* to top- k query processing and propose the *k -regret query*. The advantage of this query type is that it does not require query users to specify their utility functions. Instead, it can find the subset \mathcal{S} that minimizes the *maximum regret ratio* for a whole family of utility functions. Nanongkai et al. propose the *CUBE* algorithm to process the k -regret query for a family of linear summation utility functions, i.e., each utility function f is in the form of $f(p_i) = \sum_{j=1}^d \alpha_j \cdot p_i.c_j$ where α_j denotes the *weight* of dimension j . The CUBE algorithm is efficient, but the maximum regret ratio it obtains is quite large in practice. To obtain a smaller maximum regret ratio, in a different paper [13], Nanongkai et al. propose an interactive algorithm where query users are involved to guide the search for answers with smaller regret ratios. Peng and Wong [17] advance the k -regret query studies by utilizing geometric properties to improve the query efficiency. Chester et al. [5] also consider linear summation utility functions, and propose to compute the *k -regret minimizing sets*, which is NP-hard.

Kessler Faulkner et al. [10] build on top of the CUBE algorithm and propose three algorithms, *MinWidth*, *Area-Greedy*, and *Angle*. These three algorithms can process k -regret queries with the “concave”, “convex”, and CES utility functions. Nevertheless, the “concave” and “convex” utility functions considered are restricted to *additive forms* (See Braziunas and Boutilier [2] and Keeney and Raiffa [9] for more details on *additive utilities* and *additive independence*). They are in fact summations over a set of concave and convex functions. The CES utility functions also sum over a set of terms. In this paper, we move the study on k -regret queries from additive utility functions to multiplicative utility functions. We present an algorithm that can produce answers with bounded maximum regret ratios for k -regret queries with multiplicative utility functions. As a by-product, we also obtain a new upper bound on the maximum regret ratio for the CES utility functions which is tighter than a previously obtained upper bound [10].

Zeighami and Wong [26] propose to compute the *average regret ratio*. They do not assume any particular type of utility functions, but use sampling to obtain a few utility functions for the computation. This study is less relevant to our work and is not discussed further.

Table 2: Frequently Used Symbols

Symbol	Description
\mathcal{D}	A database
n	The cardinality of \mathcal{D}
d	The dimensionality of \mathcal{D}
k	The k -regret query parameter
\mathcal{S}	A size- k subset selected from \mathcal{D}
p_i	A point in \mathcal{D}
$p_i.c_j$	The coordinate of p_i in dimension j
t	The number of intervals that the data domain is partitioned into in each dimension

3. PRELIMINARIES

We present basic concepts and a problem definition in this section. The symbols frequently used in the discussion are summarized in Table 2.

We consider a database \mathcal{D} of n data objects. Every data object $p_i \in \mathcal{D}$ is a d -dimensional point in \mathbb{R}_+^d , where d is a positive integer and the coordinates of the points are all positive numbers. We use $p_i.c_j$ to denote the coordinate of p_i in dimension j . This coordinate represents the *utility* of the data object in dimension j . A larger value of $p_i.c_j$ denotes a larger utility and hence p_i is more preferable in dimension j . A query parameter k is given. It specifies the size of the answer set \mathcal{S} ($\mathcal{S} \subseteq \mathcal{D}$) to be returned by a k -regret query. We assume $d \leq k \leq n$.

Let $f : \mathcal{D} \rightarrow \mathbb{R}_+$ be a function that models the utility of a data object, i.e., how preferable the data object is by a query user. The *gain* of a query user over a set \mathcal{S} models the utility of the set to the query user.

DEFINITION 1 (GAIN). *Given a utility function f of a query user and a subset of data objects $\mathcal{S} \subseteq \mathcal{D}$, the gain of the query user over \mathcal{S} , $\text{gain}(\mathcal{S}, f)$, is defined as the maximum utility of any object in \mathcal{S} , i.e.,*

$$\text{gain}(\mathcal{S}, f) = \max_{p_i \in \mathcal{S}} f(p_i)$$

Continue with the example shown in Table 1. If $\mathcal{S} = \{p_1, p_3, p_5\}$ and $f_3(p_i) = p_i.c_1^{0.5} \cdot p_i.c_2^{0.5}$, then $\text{gain}(\mathcal{S}, f_3) = \max_{p_i \in \mathcal{S}} f_3(p_i) = f_3(p_1) = 2.3^{0.5} \times 80^{0.5} \approx 13.56$.

By definition, when $\mathcal{S} = \mathcal{D}$, the gain of the query user is maximized. However, the entire database \mathcal{D} is usually too large to be returned to or fully explored by the query user. When $\mathcal{S} \subset \mathcal{D}$, the query user may potentially suffer from “losing” some gain comparing with the case where $\mathcal{S} = \mathcal{D}$. This loss of gain models how much the query user may *regret* if she selects the local optimal object from \mathcal{S} and is later shown the overall optimal object in \mathcal{D} .

DEFINITION 2 (REGRET). *Given the utility function f of a query user and a subset of data objects $\mathcal{S} \subseteq \mathcal{D}$, the regret of the query user if she selects the optimal object from \mathcal{S} , $\text{regret}_{\mathcal{D}}(\mathcal{S}, f)$, is defined as the difference between the gain of the user over \mathcal{D} and her gain over \mathcal{S} , i.e.,*

$$\text{regret}_{\mathcal{D}}(\mathcal{S}, f) = \text{gain}(\mathcal{D}, f) - \text{gain}(\mathcal{S}, f)$$

The *regret ratio* is a relative measure of the regret.

DEFINITION 3 (REGRET RATIO). *Given the utility function f of a query user and a subset of data objects $\mathcal{S} \subseteq \mathcal{D}$, the regret ratio of the query user if she selects the optimal*

object from \mathcal{S} , $r_ratio_{\mathcal{D}}(\mathcal{S}, f)$, is defined as $regret_{\mathcal{D}}(\mathcal{S}, f)$ over $gain(\mathcal{D}, f)$, i.e.,

$$\begin{aligned} r_ratio_{\mathcal{D}}(\mathcal{S}, f) &= \frac{regret_{\mathcal{D}}(\mathcal{S}, f)}{gain(\mathcal{D}, f)} = \frac{regret_{\mathcal{D}}(\mathcal{S}, f)}{regret_{\mathcal{D}}(\mathcal{S}, f) + gain(\mathcal{S}, f)} \\ &= \frac{regret_{\mathcal{D}}(\mathcal{S}, f)}{regret_{\mathcal{D}}(\mathcal{S}, f) + \max_{p_j \in \mathcal{S}} f(p_j)} \\ &= \frac{\max_{p_i \in \mathcal{D}} f(p_i) - \max_{p_j \in \mathcal{S}} f(p_j)}{\max_{p_i \in \mathcal{D}} f(p_i)} \end{aligned}$$

When $\mathcal{S} = \{p_1, p_3, p_5\}$ and $f_3(p_i) = p_i.c_1^{0.5} \cdot p_i.c_2^{0.5}$, we have $gain(\mathcal{S}, f_3) = gain(\mathcal{D}, f_3) = f_3(p_1) \approx 13.56$. Thus, $regret_{\mathcal{D}}(\mathcal{S}, f_3) = 0$, and $r_ratio_{\mathcal{D}}(\mathcal{S}, f_3) = 0\%$. For a different utility function $f_4(p_i) = p_i.c_1^{0.99} \cdot p_i.c_2^{0.01}$, $gain(\mathcal{S}, f_4) = f_4(p_3) \approx 2.88$ and $gain(\mathcal{D}, f_4) = f_4(p_6) \approx 3.09$. Thus, $regret_{\mathcal{D}}(\mathcal{S}, f_4) \approx 0.21$, and $r_ratio_{\mathcal{D}}(\mathcal{S}, f_4) \approx \frac{0.21}{3.09} \approx 6.80\%$.

Given a family of utility functions \mathcal{F} , the *maximum regret ratio* formulates how regretful a query user can possibly be if she uses any utility function in \mathcal{F} and is given the set \mathcal{S} .

DEFINITION 4 (MAXIMUM REGRET RATIO). *Given a family of utility functions \mathcal{F} and a subset of data objects $\mathcal{S} \subseteq \mathcal{D}$, the maximum regret ratio if a query user with any utility function in \mathcal{F} selects the optimal object in \mathcal{S} , $mr_ratio_{\mathcal{D}}(\mathcal{S}, \mathcal{F})$, is defined as the supremum of the regret ratio of any utility function, i.e.,*

$$\begin{aligned} mr_ratio_{\mathcal{D}}(\mathcal{S}, \mathcal{F}) &= \sup_{f \in \mathcal{F}} r_ratio_{\mathcal{D}}(\mathcal{S}, f) \\ &= \sup_{f \in \mathcal{F}} \frac{\max_{p_i \in \mathcal{D}} f(p_i) - \max_{p_j \in \mathcal{S}} f(p_j)}{\max_{p_i \in \mathcal{D}} f(p_i)} \end{aligned}$$

Here, the supremum is used instead of the maximum because the family of utility functions \mathcal{F} may be infinite.

Continue with the example above. If $\mathcal{F} = \{f_3, f_4\}$ then $mr_ratio_{\mathcal{D}}(\mathcal{S}, \mathcal{F}) = \max\{0\%, 6.80\%\} = 6.80\%$.

The k -regret query aims to return the size- k subset $\mathcal{S} \subseteq \mathcal{D}$ that *minimizes* the maximum regret ratio.

DEFINITION 5 (K-REGRET QUERY). *Given a family of utility functions \mathcal{F} , the k -regret query returns a size- k subset $\mathcal{S} \subseteq \mathcal{D}$, such that the maximum regret ratio over \mathcal{S} is smaller than or equal to that over any other size- k subset $\mathcal{S}' \subseteq \mathcal{D}$. Formally,*

$$\forall \mathcal{S}' \subseteq \mathcal{D} \cap |\mathcal{S}'| = k : mr_ratio_{\mathcal{D}}(\mathcal{S}, \mathcal{F}) \leq mr_ratio_{\mathcal{D}}(\mathcal{S}', \mathcal{F})$$

Specific utility functions are not always available because the query users are not usually pre-known and their utility functions may not be specified precisely. The k -regret query does not require any specific utility functions to be given. Instead, the query considers a family of functions of a certain form such as the linear functions [14], i.e., $f(p_i) = \sum_{j=1}^d \alpha_j \cdot p_i.c_j$ where α_j is the *weight* of dimension j . The k -regret query minimizes the maximum regret ratio of any utility function of such form (without knowing the value of α_j).

Our contribution to the study of k -regret queries is the consideration of a family of *multiplicative utility functions* (MUFs).

DEFINITION 6 (MULTIPLICATIVE UTILITY FUNCTION). *We define a multiplicative utility function (MUF) f to be a utility function of the following form:*

$$f(p_i) = \prod_{j=1}^n p_i.c_j^{\alpha_j},$$

where $\alpha_j \geq 0$ is a function parameter and $\sum_{j=1}^d \alpha_j \leq 1$.

DEFINITION 7 (K-REGRET QUERY WITH MUFs). *The k -regret query with MUFs takes a database \mathcal{D} of d -dimensional points and a set \mathcal{F} of MUFs as the input, where the parameters values $\{\alpha_1, \alpha_2, \dots, \alpha_d\}$ of each MUF may be different. The query returns a size- k subset $\mathcal{S} \subseteq \mathcal{D}$, such that the maximum regret ratio $mr_ratio_{\mathcal{D}}(\mathcal{S}, \mathcal{F})$ is minimized.*

We assume that $p_i.c_j$ has been normalized into the range of $(1, 2]$ to simplify the derivation of the maximum regret ratio bounds. This can be done by a normalization function $\mathcal{N}(p_i.c_j) = 1 + \frac{p_i.c_j}{\max_{p_i \in \mathcal{D}} \{p_i.c_j\}}$. In fact, it is common to normalize the data domain in different dimensions into the same range, so that utility values of different dimensions become more comparable. Note that our derivation of the bounds still holds without this assumption, although the bounds may become less concise.

Scale invariance. It has been shown [10, 14] that k -regret queries with additive utility functions are *scale invariant*, i.e., scaling the data domain in any dimension does not change the maximum regret ratio of a set \mathcal{S} . This property is preserved in k -regret queries with MUFs. For an MUF $f(p_i) = \prod_{d=1}^n p_i.c_j^{\alpha_j}$, we can scale each dimension by a factor $\lambda_j > 0$, resulting in a new MUF $f'(p_i) = \prod_{d=1}^n (\lambda_j \cdot p_i.c_j)^{\alpha_j} = \prod_{d=1}^n \lambda_j^{\alpha_j} \cdot \prod_{d=1}^n p_i.c_j^{\alpha_j} = (\prod_{d=1}^n \lambda_j^{\alpha_j}) f(p_i)$. Such scaling does not affect the regret ratio (and hence the maximum regret ratio), i.e., $r_ratio_{\mathcal{D}}(\mathcal{S}, f') = r_ratio_{\mathcal{D}}(\mathcal{S}, f)$:

$$\begin{aligned} r_ratio_{\mathcal{D}}(\mathcal{S}, f') &= \frac{\max_{p_i \in \mathcal{D}} f'(p_i) - \max_{p_j \in \mathcal{S}} f'(p_j)}{\max_{p_i \in \mathcal{D}} f'(p_i)} \\ &= \frac{(\prod_{d=1}^n \lambda_j^{\alpha_j})(\max_{p_i \in \mathcal{D}} f(p_i) - \max_{p_j \in \mathcal{S}} f(p_j))}{(\prod_{d=1}^n \lambda_j^{\alpha_j}) \max_{p_i \in \mathcal{D}} f(p_i)} \\ &= \frac{\max_{p_i \in \mathcal{D}} f(p_i) - \max_{p_j \in \mathcal{S}} f(p_j)}{\max_{p_i \in \mathcal{D}} f(p_i)} = r_ratio_{\mathcal{D}}(\mathcal{S}, f) \end{aligned}$$

In what follows, for conciseness, we refer to the regret, the regret ratio, and the maximum regret ratio of a query user simply as the regret, the regret ratio, and the maximum regret ratio, respectively.

4. THE K-REGRET QUERY WITH MULTIPLICATIVE UTILITY FUNCTIONS

We propose an algorithm named *MinVar* to process k -regret queries with MUFs in Section 4.1. We derive the bounds of the maximum regret ratios of the query answers returned by MinVar in Sections 4.2 and 4.3. We further improve the maximum regret ratios of the query answers through a heuristic based algorithm in Section 4.4.

4.1 The MinVar Algorithm

MinVar shares a similar overall algorithmic approach with that of CUBE [14] and MinWidth [10] which were proposed to process k -regret queries with additive utility functions.

As summarized in Algorithm 1, MinVar first finds the optimal point p_i^* in dimension i for the first $d-1$ dimensions, i.e., $p_i^*.c_i$ is the largest utility in dimension i ($i = 1, 2, \dots, d-1$). These $d-1$ points are added to \mathcal{S} (Lines 1 to 5). Then, the algorithm partitions each dimension i of the data domain into $t = \lfloor (k-d+1)^{\frac{1}{d-1}} \rfloor$ intervals for the first $d-1$

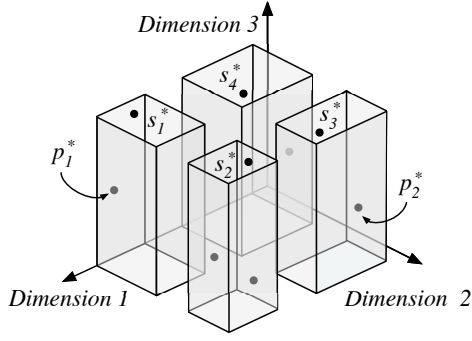


Figure 1: The MinVar algorithm

dimensions (Lines 6 to 8). Here, the value of t is chosen so that we can obtain sufficient points to be added to \mathcal{S} to create a size- k subset. The t intervals in each of the first $d-1$ dimensions together partition the d dimensional data space into t^{d-1} buckets. The algorithm selects one point s^* in each bucket that has the largest utility $s^*.c_d$ in dimension d , and adds s^* to \mathcal{S} (Lines 9 to 12). There are $t^{d-1} = k - d + 1$ points added in this step. Together with the $d-1$ points previously added, \mathcal{S} now has k points, which are then returned as the query answer (Line 13).

Figure 1 gives an example. Suppose $d = 3$ and $k = 6$, then $t = \lfloor (6 - 3 + 1)^{\frac{1}{3-1}} \rfloor = 2$. We first add the two points p_1^* and p_2^* to \mathcal{S} which have the largest utility in dimensions 1 and 2, respectively. Then the data domain in dimensions 1 and 2 are each partitioned into $t = 2$ intervals, forming $t^{d-1} = 2^{3-1} = 4$ buckets in the data space. Four more points s_1^*, s_2^*, s_3^* , and s_4^* are added to \mathcal{S} , each from a different bucket and is the point with the largest utility in dimension 3 in that bucket.

Algorithm 1: MinVar

Input: $\mathcal{D} = \{p_1, p_2, \dots, p_n\}$: a d -dimensional database; k : the size of the answer set.
Output: \mathcal{S} : a size- k subset of \mathcal{D} .

```

1  $\mathcal{S} \leftarrow \emptyset$ ;
2 for  $i = 1, 2, \dots, d-1$  do
3   Find  $p_i^*$  which has the largest utility  $p_i^*.c_i$  in dimension  $i$ ;
4    $c_i^T \leftarrow p_i^*.c_i$ ;
5    $\mathcal{S} \leftarrow \mathcal{S} \cup \{p_i^*\}$ ;
6  $t \leftarrow \lfloor (k - d + 1)^{\frac{1}{d-1}} \rfloor$ ;
7 for  $i = 1, 2, \dots, d-1$  do
8    $bps[i] \leftarrow \text{FindBreakpoints}(\mathcal{D}, t, n, i, c_i^T)$ ;
9 for each  $(d-1)$ -integer combination
10   $1 \leq j_1 \leq t, 1 \leq j_2 \leq t, \dots, 1 \leq j_{d-1} \leq t$  do
11     $B \leftarrow \{p \in \mathcal{D} \mid \forall i \in [1..d-1] : bps[i][j_i].lo \leq p.c_i \leq bps[i][j_i].hi\}$ ;
12     $s^* \leftarrow \argmax_{p \in B} p.c_d$ ;
13     $\mathcal{S} \leftarrow \mathcal{S} \cup \{s^*\}$ ;
14 return  $\mathcal{S}$ ;
```

The FindBreakpoints algorithm. Our contribution of MinVar lies in the sub-algorithm *FindBreakpoints* to find the breakpoints to partition each dimension i of the data domain into t intervals (Line 8). The intuition of the algorithm is as follows. The optimal point p^* for any utility function f must lie in one of the buckets created by MinVar. Let this bucket be \mathcal{B} . The algorithm selects a point s^* from \mathcal{B} to represent this bucket and adds it to \mathcal{S} . If p^* is selected to be s^* , then the regret ratio is 0. To maximize the worst-case probability of p^* being selected, we should partition dimen-

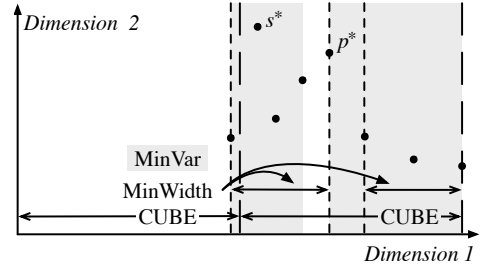


Figure 2: Creating the intervals in a dimension

sion i such that each interval contains the same number of points. Otherwise, if the intervals are skewed and p^* lies in a large interval, its probability of being selected is small.

In CUBE [14], the data domain is simply broken evenly, i.e., each interval has the same size of $\frac{c_i^T}{t}$, where c_i^T denotes the largest utility in dimension i (assuming that the data domain starts at 0). When the data points are not uniformly distributed, the probability of p^* being selected to represent its bucket is small. Figure 2 gives an example where $d = 2$. The two intervals created by CUBE in dimension 1 are highly unbalanced in the number of points in each interval. The point p^* has a large utility in both dimensions and may be optimal for many MUFs. However, it will not be selected by CUBE to represent its bucket, since it falls in a dense bucket and there is another point s^* with a larger utility in dimension 2. In MinWidth [10], the data domain is broken with a greedy heuristic. This heuristic leaves out some *empty intervals* with no data points, and uses a binary search to determine the minimal interval width such that t intervals of such width can cover the rest of the data domain. This algorithm handles sparse data better, but the equi-width intervals still do not handle skewed data well. Figure 2 shows two intervals created by MinWidth which are still unbalanced (one has 5 points and the other has 3). The two points p^* and s^* are still in the same bucket.

Algorithm 2: FindBreakpoints

Input: $\mathcal{D} = \{p_1, p_2, \dots, p_n\}$: a d -dimensional database; t : number of intervals; n : size of \mathcal{D} ; i : dimension number for which the breakpoints are to be computed; c_i^T : the largest utility in dimension i .
Output: $bps[i]$: an array of t pairs of breakpoints.

```

1 Sort  $\mathcal{D}$  ascendingly on dimension  $i$ ; let the sorted point sequence be  $p'_1, p'_2, \dots, p'_n$ ;
2  $hi \leftarrow 0, \delta \leftarrow 0$ ;
3 while  $hi \neq n$  do
4    $lo \leftarrow 1$ ;
5   for  $j = 1, 2, \dots, t$  do
6      $hi' \leftarrow \min\{lo + \lceil n/t \rceil - 1 + \delta, n\}$ ;
7     Find the largest  $hi \in [lo..hi']$  such that
8        $p'_{hi}.c_i - p'_{lo}.c_i \leq \frac{c_i^T}{t}$ ;
9      $bps[i][j].lo \leftarrow p'_{lo}.c_i$ ;
10     $bps[i][j].hi \leftarrow p'_{hi}.c_i$ ;
11     $lo \leftarrow hi + 1$ ;
12   $\delta \leftarrow \delta + inc$ ;
13 return  $bps[i]$ ;
```

To overcome these limitations, our FindBreakpoints algorithm adaptively uses t variable-width intervals such that the number of points in each interval is as close to $\lceil \frac{n}{t} \rceil$ as possible (cf. Figure 2, where the two gray intervals created

by MinVar contain $\lceil \frac{8}{2} \rceil = 4$ points each; p^* will be selected to represent its bucket). To help derive the maximum regret ratio bounds in the following subsections, we also require that the width of each interval does not exceed $\frac{c_i^\tau}{t}$.¹ Under this constraint it is not always possible to create t intervals with exactly $\lceil \frac{n}{t} \rceil$ points in each interval. We allow $\lceil \frac{n}{t} \rceil + \delta$ data points in each interval, where δ is a parameter that will be adaptively chosen by the algorithm. At start $\delta = 0$.

Algorithm 2 summarizes the FindBreakpoints algorithm. This algorithm first sorts the data points in ascending order of their coordinates in dimension i . The sorted points are denoted as p'_1, p'_2, \dots, p'_n (Line 1). The algorithm then creates t intervals, where lo and hi represent the subscript lower and upper bounds of the data points to be put into one interval, respectively. At start, $lo = 1$ (Line 4). Between lo and $lo + \lceil \frac{n}{t} \rceil - 1 + \delta$, the algorithm finds the largest subscript hi such that $p'_{hi}.c_i - p'_{lo}.c_i$ is bounded by $\frac{c_i^\tau}{t}$. Then we have obtained the two breakpoints of the first interval $bps[i][1].lo = p'_{lo}.c_i$ and $bps[i][1].hi = p'_{hi}.c_i$, where $bps[i]$ is an array to store the intervals in dimension i . We update lo to be $hi + 1$, and repeat the process above to create the next interval (Lines 5 to 10). When t intervals are created, if they cover all the n points, we have successfully created the intervals for dimension i . Otherwise, we need to allow a larger number of points in one interval. We increase δ by inc which is a system parameter (Line 11), and repeat the above procedure to create t intervals until n points are covered. Then we return the interval array $bps[i]$ (Line 12). Note that the algorithm always terminates, because when δ increases to n , the algorithm will simply create intervals with width $\frac{c_i^\tau}{t}$. The t intervals must cover the entire data domain and hence cover all n points.

Complexity. FindBreakpoints uses a database \mathcal{D} of n points and an array $bps[i]$ to store t intervals. The space complexity is $O(n + t)$ where $t = \lfloor (k - d + 1)^{\frac{1}{d-1}} \rfloor$ is usually small. The inner loop of FindBreakpoints (Lines 5 to 10) has t iterations. In each iteration, computing hi requires a binary search between p'_{lo} and $p'_{hi'}$, which takes $O(\log n)$ time. Thus, The inner loop takes $O(t \log n)$ time. The outer loop has $\frac{n}{inc}$ iterations in the worst case. Together, FindBreakpoints takes $O(\frac{tn \log n}{inc})$ time.

MinVar uses a database \mathcal{D} of n points, an answer set \mathcal{S} of size k , a $(d-1) \times t$ two dimensional array bps . An array of size $t^{d-1} = k - d + 1$ is also needed to help select the points s^* in the $k - d + 1$ buckets. The space complexity is $O(n + k + dt)$. The first loop of MinVar (Lines 2 to 5) takes $O(nd)$ time. The second loop (Lines 7 and 8) calls FindBreakpoints for $d - 1$ times, which takes $O(\frac{tdn \log n}{inc})$ time. The third loop (Lines 9 to 12) finds a point s^* in each of the $k - d + 1$ buckets. A linear scan on the database \mathcal{D} is needed for this task. For each point p visited, we need a binary search on each of the $d - 1$ arrays $bps[i]$ to identify the bucket of p , and to update the point selected s^* in that bucket if

¹It should be $\frac{c_i^\tau - 1}{t}$ to be exact where 1 is the lower bound of the data domain. We write $\frac{c_i^\tau}{t}$ here to keep it consistent with CUBE and to ease the discussion.

needed. This takes $O(nd \log t)$ time. Overall, MinVar takes $O(nd + \frac{tdn \log n}{inc} + nd \log t)$ time. Here, $\frac{n}{inc}$ is a controllable parameter of the system. In the experiments, we set $inc = 0.01\%$. The time complexity then simplifies to $O(nd \log t)$.

4.2 Upper Bound

We derive an upper bound for the maximum regret ratio of a set \mathcal{S} returned by MinVar.

THEOREM 1. Let $\mathcal{F} = \{f | f(p_i) = \prod_{j=1}^d p_i.c_j^{\alpha_j}\}$ be a set of MUFs, where $\alpha_j \geq 0$, $\sum_{j=1}^d \alpha_j \leq 1$, and $1 \leq p_i.c_j \leq 2$. The maximum regret ratio $mr_ratio_{\mathcal{D}}(\mathcal{S}, \mathcal{F})$ of an answer set \mathcal{S} returned by MinVar satisfies

$$mr_ratio_{\mathcal{D}}(\mathcal{S}, \mathcal{F}) \leq \ln(1 + \frac{1}{t})$$

Here, $t = \lfloor (k - d + 1)^{\frac{1}{d-1}} \rfloor$.

PROOF. We prove the theorem by showing that for each function $f \in \mathcal{F}$, the regret ratio $r_ratio_{\mathcal{D}}(\mathcal{S}, f) \leq \ln(1 + \frac{1}{t})$, which means that the maximum regret ratio of \mathcal{F} must be less than or equal to $\ln(1 + \frac{1}{t})$.

Let p^* be the point in \mathcal{D} with the largest utility computed by f , i.e.,

$$p^* = \operatorname{argmax}_{p_i \in \mathcal{D}} f(p_i)$$

Let s^* be the point in \mathcal{S} that is selected by MinVar in the same bucket where p^* lies in. We have:

$$\begin{aligned} regret_{\mathcal{D}}(\mathcal{S}, f) &= \max_{p_i \in \mathcal{D}} f(p_i) - \max_{p_i \in \mathcal{S}} f(p_i) \\ &\leq f(p^*) - f(s^*) \\ &= \prod_{j=1}^d p^*.c_j^{\alpha_j} - \prod_{j=1}^d s^*.c_j^{\alpha_j} \\ &= \exp(\ln \prod_{j=1}^d p^*.c_j^{\alpha_j}) - \exp(\ln \prod_{j=1}^d s^*.c_j^{\alpha_j}) \end{aligned}$$

Since $g(x) = e^x$ is convex, based on Lagrange's Mean Value Theorem, we have $e^x - e^y \leq (x - y) \cdot e^x$ where $x \geq y$. Thus,

$$\begin{aligned} regret_{\mathcal{D}}(\mathcal{S}, f) &\leq (\ln \prod_{j=1}^d p^*.c_j^{\alpha_j} - \ln \prod_{j=1}^d s^*.c_j^{\alpha_j}) \cdot \prod_{j=1}^d p^*.c_j^{\alpha_j} \\ &= (\sum_{j=1}^d \alpha_j \ln p^*.c_j - \sum_{j=1}^d \alpha_j \ln s^*.c_j) \cdot \prod_{j=1}^d p^*.c_j^{\alpha_j} \\ &= \left[\sum_{j=1}^d \alpha_j (\ln p^*.c_j - \ln s^*.c_j) \right] \cdot \prod_{j=1}^d p^*.c_j^{\alpha_j} \end{aligned}$$

Since MinVar selects the point in a bucket with the largest value in dimension d , we know that $p^*.c_d \leq s^*.c_d$ and hence $\ln p^*.c_d \leq \ln s^*.c_d$, i.e., $\ln p^*.c_d - \ln s^*.c_d \leq 0$. Thus, we can remove the utility in dimension d from the computation and relax the regret to be:

$$\begin{aligned} regret_{\mathcal{D}}(\mathcal{S}, f) &\leq \left[\sum_{j=1}^{d-1} \alpha_j (\ln p^*.c_j - \ln s^*.c_j) \right] \cdot \prod_{j=1}^d p^*.c_j^{\alpha_j} \\ &= \left[\sum_{j=1}^{d-1} \alpha_j \ln \left(\frac{p^*.c_j}{s^*.c_j} \right) \right] \cdot \prod_{j=1}^d p^*.c_j^{\alpha_j} \end{aligned}$$

Since s^* is selected from the same bucket where p^* lies in, $p^*.c_j - s^*.c_j$ must be constrained by the bucket size in dimension j , which is $\frac{c_j^\tau - 1}{t}$ where c_j^τ and 1 are the largest and smallest utility values in dimension j , i.e.,

$$\forall j \in [1..d-1], p^*.c_j - s^*.c_j \leq \frac{c_j^\tau - 1}{t}$$

Thus,

$$\frac{p^*.c_j}{s^*.c_j} \leq 1 + \frac{c_j^\tau - 1}{t \cdot s^*.c_j}$$

Since $1 < s^*.c_j \leq c_j^\tau \leq 2$, we have

$$\frac{p^*.c_j}{s^*.c_j} < 1 + \frac{1}{t}$$

Therefore,

$$\text{regret}_{\mathcal{D}}(\mathcal{S}, f) < \left[\sum_{j=1}^{d-1} \alpha_j \ln(1 + \frac{1}{t}) \right] \cdot \prod_{j=1}^d p^*.c_j^{\alpha_j}$$

For the regret ratio $r_ratio_{\mathcal{D}}(\mathcal{S}, f)$ we have

$$\begin{aligned} r_ratio_{\mathcal{D}}(\mathcal{S}, f) &= \frac{\text{regret}_{\mathcal{D}}(\mathcal{S}, f)}{\text{gain}(\mathcal{D}, f)} \\ &< \frac{\left[\sum_{j=1}^{d-1} \alpha_j \ln(1 + \frac{1}{t}) \right] \cdot \prod_{j=1}^d p^*.c_j^{\alpha_j}}{\prod_{j=1}^d p^*.c_j^{\alpha_j}} \\ &= \sum_{j=1}^{d-1} \alpha_j \ln(1 + \frac{1}{t}) = \ln(1 + \frac{1}{t})^{\sum_{j=1}^{d-1} \alpha_j} \\ &\leq \ln(1 + \frac{1}{t}) \end{aligned}$$

□

In the theorem, $t = \lfloor (k - d + 1)^{\frac{1}{d-1}} \rfloor$. This means that as k increases, the maximum regret ratio is expected to decrease; when d increases, the maximum regret ratio is expected to increase. Intuitively, if we return more points (a larger k), then the probability of returning the points that better satisfy the utility functions is higher, and hence the regret ratios would be smaller. If there are more dimensions (a larger d), then more regret may be accumulated over the dimensions, and hence the regret ratios would be larger. This shows that the upper bound obtained has a good explanatory power. For simplicity, we say that the upper bound grows in a scale of $O(\ln(1 + \frac{1}{k^{\frac{1}{d-1}}}))$.

To give an example, consider $\mathcal{F} = \{f_3, f_4\}$ where $f_3(p_i) = p_i.c_1^{0.5} \cdot p_i.c_2^{0.5}$ and $f_4(p_i) = p_i.c_1^{0.99} \cdot p_i.c_2^{0.01}$. Then $d = 2$. Let $k = 3$, which means $t = 2$. The upper bound of the maximum regret ratio $\ln(1 + \frac{1}{t}) = \ln \frac{3}{2} \approx 40.54\%$. As k increases (e.g., to 20), this upper bound will decrease (e.g., to $\ln \frac{20}{19} \approx 5.13\%$).

4.3 Lower Bound

We derive a lower bound of the maximum regret ratio by showing that, given a family of MUFs \mathcal{F} , it is impossible to bound the regret ratio to below $\Omega(\frac{1}{k^2})$ for a database \mathcal{D} of 2-dimensional points (i.e., $d = 2$).

THEOREM 2. *Given $k > 0$, there must be a database \mathcal{D} of 2-dimensional points such that the maximum regret ratio*

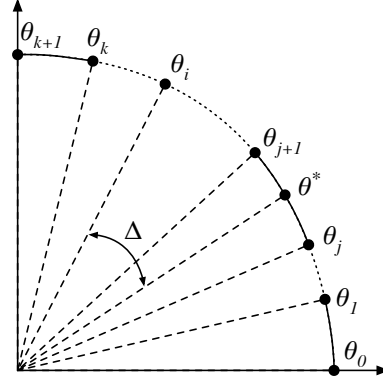


Figure 3: Lower bound illustration

of any size- k subset $\mathcal{S} \subseteq \mathcal{D}$ over a family of MUFs \mathcal{F} is at least $\Omega(\frac{1}{k^2})$.

PROOF. We assume a data space of $(1, e] \times (1, e]$ in this proof. Consider an infinite set \mathcal{D} of 2-dimensional points, where each point p satisfies

$$\begin{cases} p.c_1 = e^{\cos \theta} \\ p.c_2 = e^{\sin \theta} \end{cases} \quad 0 < \theta \leq \frac{\pi}{2}$$

Given a size- k subset $\mathcal{S} \subseteq \mathcal{D}$, each point $s_i \in \mathcal{S}$ corresponds to a $\theta_i \in (0, \frac{\pi}{2}]$ where $s_i.c_1 = e^{\cos \theta_i}$ and $s_i.c_2 = e^{\sin \theta_i}$. Assume that the points s_1, s_2, \dots, s_k in \mathcal{S} are sorted in ascending order of their corresponding θ_i values, i.e., $0 < \theta_1 \leq \theta_2 \leq \dots \leq \theta_k \leq \frac{\pi}{2}$. Further, let $\theta_0 = 0$ and $\theta_{k+1} = \frac{\pi}{2}$. Consider a polar coordinate system as illustrated in Figure 3. Then θ_i ($i \in [0..k+1]$) can be represented as a point on a unit circle. Based on the pigeonhole principle, there must be some j ($j \in [0..k]$) such that

$$\theta_{j+1} - \theta_j \geq \frac{\pi}{2(k+1)}$$

Let θ^* be in the middle of θ_j and θ_{j+1} , i.e.,

$$\theta^* = \frac{\theta_j + \theta_{j+1}}{2}$$

We construct an MUF f where the optimal point p^* corresponds to θ^* , i.e., $p^*.c_1 = e^{\cos \theta^*}$ and $p^*.c_2 = e^{\sin \theta^*}$, and prove the theorem based on the regret ratio of f .

Consider an MUF $f(p) = p.c_1^{\cos \theta^*} \cdot p.c_2^{\sin \theta^*}$.

$$\begin{aligned} \ln f(p) &= \ln(p.c_1^{\cos \theta^*} \cdot p.c_2^{\sin \theta^*}) \\ &= \cos \theta^* \cdot \ln p.c_1 + \sin \theta^* \cdot \ln p.c_2 \\ &= \cos \theta^* \cdot \cos \theta + \sin \theta^* \cdot \sin \theta \end{aligned}$$

Let $g(\theta) = \cos \theta^* \cdot \cos \theta + \sin \theta^* \cdot \sin \theta$. By letting $g'(\theta) = 0$ we obtain $\theta = \theta^*$, which means that $\ln f(p)$ is maximum when $\theta = \theta^*$, and $f(p)$ is maximum when $p = p^*$.

$$\ln f(p^*) = \cos^2 \theta^* + \sin^2 \theta^* = 1; \quad f(p^*) = e.$$

Meanwhile, let s_i be the optimal point for f in \mathcal{S} . Since there is no other points in \mathcal{S} that is between θ_j and θ_{j+1} ,

$$|\theta_i - \theta^*| = \Delta \geq \theta_{j+1} - \theta^* = \theta^* - \theta_j \geq \frac{\pi}{4(k+1)}$$

We consider the case where $\theta_i - \theta^* = \Delta$. The other case where $\theta^* - \theta_i = \Delta$ is symmetric. We omit it for conciseness.

$$\begin{aligned} \ln f(s_i) &= \ln(s_i \cdot c_1^{\cos \theta^*} \cdot s_i \cdot c_2^{\sin \theta^*}) \\ &= \cos(\theta^* + \Delta) \cdot \cos \theta^* + \sin(\theta^* + \Delta) \cdot \sin \theta^* \\ &= (\cos \theta^* \cos \Delta - \sin \theta^* \sin \Delta) \cdot \cos \theta^* + \\ &\quad (\sin \theta^* \cos \Delta + \cos \theta^* \sin \Delta) \cdot \sin \theta^* \\ &= \cos^2 \theta^* \cdot \cos \Delta + \sin^2 \theta^* \cdot \cos \Delta \\ &= \cos \Delta \end{aligned}$$

Thus, $f(s_i) = e^{\cos \Delta}$, and $r_ratio_{\mathcal{D}}(\mathcal{S}, f)$ satisfies

$$r_ratio_{\mathcal{D}}(\mathcal{S}, f) = \frac{f(p^*) - f(s_i)}{f(p^*)} = \frac{e - e^{\cos \Delta}}{e} = 1 - e^{\cos \Delta - 1}$$

Based on the Maclaurin series, we have

$$e^{\cos \Delta - 1} = 1 - \frac{\Delta^2}{2} + \frac{\Delta^4}{6} - \dots$$

Thus,

$$r_ratio_{\mathcal{D}}(\mathcal{S}, f) = \frac{\Delta^2}{2} - \frac{\Delta^4}{6} + \dots$$

We already know that

$$\Delta \geq \frac{\pi}{4(k+1)}$$

Therefore,

$$r_ratio_{\mathcal{D}}(\mathcal{S}, f) \geq \frac{\pi^2}{32(k+1)^2} - o\left(\frac{1}{k^4}\right)$$

This means that $r_ratio_{\mathcal{D}}(\mathcal{S}, f)$ is at least $\Omega(\frac{1}{k^2})$. \square

4.4 Heuristic to Lower the Regret Ratio

The upper bound derived in Section 4.2 suggests that the maximum regret ratio decreases with the increase of t . This parameter decides the number of buckets from which the points in the answer set \mathcal{S} are selected. The parameter itself is determined by d and k together, which are fixed once a database \mathcal{D} and a k -regret query is given. In this subsection, we propose a heuristic to increase the value of t without changing d or k , aiming to obtain lower regret ratios.

This heuristic is based on the *dominate* relationship [1]. Let p_i and p_j be two points in a set \mathcal{S} where $i \neq j$. Point p_i is said to dominate point p_j if and only if $\forall l \in [1..d], p_i \cdot c_l \geq p_j \cdot c_l$. If p_i dominates p_j , then $f(p_i) \geq f(p_j)$ holds for any MUF f . Thus, once p_i has been added to \mathcal{S} , p_j can be discarded. We call p_j a *redundant point*. Based on this observation, we modify the MinVar algorithm to remove the redundant points in \mathcal{S} , aiming to obtain a redundant free set \mathcal{S} . We call this modified algorithm *RF-MinVar*.

As summarized in Algorithm 3, RF-MinVar uses the same procedure as that of MinVar to compute an answer set \mathcal{S} based on a given t (Lines 1 to 5 and 8 to 14). After that, RF-MinVar removes the redundant points from \mathcal{S} (Line 15). This is done by a simple scan to check for any points being dominated. When the redundant points are removed, there may be a few vacancies open in \mathcal{S} . To fill up the vacancies, we wrap the procedure of computing \mathcal{S} with a loop (Line 7). In each iteration we increase the value of t by 1 (Line 8), which creates more buckets and leads to more points being added to \mathcal{S} . If $|\mathcal{S}| \geq k$ after removing the redundant points,

or a predefined number of iterations itr_{max} is reached, the loop terminates. Then, if $|\mathcal{S}| < k$, we randomly select points in \mathcal{D} to fill up \mathcal{S} (Line 17); if $|\mathcal{S}| > k$, only the first k points are kept. We then return the set \mathcal{S} (Line 18).

Algorithm 3: RF-MinVar

Input: $\mathcal{D} = \{p_1, p_2, \dots, p_n\}$: a d -dimensional database; k : the size of the answer set.
Output: \mathcal{S} : a size- k subset of \mathcal{D} .

```

1  $\mathcal{S} \leftarrow \emptyset$ ;
2 for  $i = 1, 2, \dots, d-1$  do
3   Find  $p_i^*$  which has the largest utility  $p_i^* \cdot c_i$  in dimension  $i$ ;
4    $c_i^T \leftarrow p_i^* \cdot c_i$ ;
5    $\mathcal{S} \leftarrow \mathcal{S} \cup \{p_i^*\}$ ;
6  $itr \leftarrow 0$ ;
7 while  $|\mathcal{S}| < k$  and  $itr < itr_{max}$  do
8    $t \leftarrow \lfloor (k-d+1)^{\frac{1}{d-1}} \rfloor + itr$ ;
9   for  $i = 1, 2, \dots, d-1$  do
10     $bps[i] \leftarrow FindBreakpoints(\mathcal{D}, t, n, i, c_i^T)$ ;
11   for each  $(d-1)$ -integer combination
12      $1 \leq j_1 \leq t, 1 \leq j_2 \leq t, \dots, 1 \leq j_{d-1} \leq t$  do
13        $B \leftarrow \{p \in \mathcal{D} | \forall i \in [1..d-1] : bps[i][j_i] \cdot lo \leq p \cdot c_i \leq bps[i][j_i] \cdot hi\}$ ;
14        $s^* \leftarrow argmax_{p \in B} p \cdot c_d$ ;
15        $\mathcal{S} \leftarrow \mathcal{S} \cup \{s^*\}$ ;
16   Eliminate redundant points from  $\mathcal{S}$ ;
17    $itr++$ ;
18  $\mathcal{S} \leftarrow \mathcal{S} \cup \{k - |\mathcal{S}| \text{ random points in } \mathcal{D} \text{ that are not in } \mathcal{S}\}$ ;
19 return First  $k$  points in  $\mathcal{S}$ ;
```

Discussion. RF-MinVar keeps the non-dominated points in \mathcal{S} computed when $t = \lfloor (k-d+1)^{\frac{1}{d-1}} \rfloor$. Thus, it retains the upper bound of the maximum regret ratio derived in Section 4.2. RF-MinVar can reduce the actual regret ratios obtained, as it may also return the points computed with larger t values. The extra cost to scan and remove the redundant points is minimal, as k is usually a very small number (e.g., less than 100). The main cost of RF-MinVar is in the loop to iterate through multiple values of t and compute \mathcal{S} . In the experiments, we use itr_{max} to control the maximum number of iterations. We observe that $itr_{max} = 11$ is sufficient in the data sets tested.

5. CASE STUDIES

In this section showcase the applicability of the MinVar algorithm by deriving the maximum regret ratio bounds when applying MinVar on k -regret queries with two special types of utility functions, the Cobb-Douglas function and the Constant Elasticity of Substitution (CES) function.

5.1 The K-Regret Query with Cobb-Douglas Functions

The Cobb-Douglas function was first proposed as a production function to model the relationship between multiple inputs and the amount of output generated [4]. It was later generalized [24] and used as a utility function [6].

DEFINITION 8 (COBB-DOUGLAS FUNCTION). *A generalized Cobb-Douglas function with d inputs x_1, x_2, \dots, x_d is a mapping $\mathcal{X} : \mathbb{R}_+^d \rightarrow \mathbb{R}_+$,*

$$\mathcal{X}(x_1, x_2, \dots, x_d) = A \prod_{j=1}^d x_j^{\alpha_j}$$

Here, $A > 0$ and $\alpha_j \geq 0$ are the function parameters.

The generalized Cobb-Douglas function is very similar to the MUF introduced in Section 3. The d inputs here can be seen as a data point of d dimensions and input x_j is the utility in dimension j . MinVar can process the k -regret query with Cobb-Douglas functions straightforwardly.

To derive an upper bound of the maximum regret ratio for a set of Cobb-Douglas functions $\mathcal{F} = \{\mathcal{X}_1, \mathcal{X}_2, \dots, \mathcal{X}_n\}$, we transform each function \mathcal{X}_i to an MUF by scaling the parameter A to 1. It can be shown straightforwardly that this scaling does not affect the regret ratio or the maximum regret ratio. Assuming that x_j has been normalized into the range of $(1, 2]$. Then the regret ratio upper bound derived in Section 4.2 applies to the function \mathcal{X}_i , i.e.,

$$r_ratio_{\mathcal{D}}(\mathcal{S}, \mathcal{X}_i) \leq \ln(1 + \frac{1}{t})^{\sum_{j=1}^{d-1} \alpha_j}$$

Here, each function \mathcal{X}_i has a different set of parameters $\{\alpha_1, \alpha_2, \dots, \alpha_d\}$. If $\sum_{j=1}^d \alpha_j \leq 1$ holds for every $\mathcal{X}_i \in \mathcal{F}$, then the maximum regret ratio is bounded by

$$mr_ratio_{\mathcal{D}}(\mathcal{S}, \mathcal{F}) \leq \ln(1 + \frac{1}{t})$$

Otherwise, the maximum regret ratio is bounded by

$$mr_ratio_{\mathcal{D}}(\mathcal{S}, \mathcal{F}) \leq \ln(1 + \frac{1}{t})^{\alpha^\tau},$$

$\alpha^\tau = \max\{\sum_{j=1}^{d-1} \mathcal{X}_i \cdot \alpha_j | \mathcal{X}_i \in \mathcal{F}, \mathcal{X}_i \cdot \alpha_j \text{ is a parameter of } \mathcal{X}_i\}$.

Similarly, the lower bound $\Omega(\frac{1}{k^2})$ of the maximum regret ratio derived in Section 4.3 also applies.

5.2 The K-Regret Query with CES Functions

The CES function is closely related to the Cobb-Douglas function. It is also used as a production function as well as a utility function [22, 23].

DEFINITION 9 (CES FUNCTION). A generalized CES function with d inputs x_1, x_2, \dots, x_d is a mapping $\mathcal{X} : \mathbb{R}_+^d \rightarrow \mathbb{R}_+$,

$$\mathcal{X}(x_1, x_2, \dots, x_d) = A(\sum_{j=1}^d \alpha_j x_j^\rho)^{\frac{\gamma}{\rho}}$$

Here, $A > 0$, $\alpha_j \geq 0$, $\rho < 1$ ($\rho \neq 0$), and $\gamma > 0$ are the function parameters.

When ρ approaches 0 in the limit, the CES function will become a Cobb-Douglas function.

The MinVar algorithm can also process k -regret queries with CES utility functions. To derive bounds for the maximum regret ratio, we simplify and rewrite the CES function \mathcal{X} as a function f in the following form (assuming that $A = \gamma = 1$):

$$f(p_i) = (\sum_{j=1}^d \alpha_j \cdot p_i \cdot c_j^b)^{\frac{1}{b}}$$

Here, $0 < b < 1$ and $\alpha_j \geq 0$.

It has been shown in an earlier paper [10] that the maximum regret ratio for k -regret queries with CES utility functions is bounded between $\Omega(\frac{1}{bk^2})$ and $O(\frac{1}{bk^{\frac{b}{d-1}}})$. The lower bound also applies to our MinVar algorithm. In what follows, we derive a new upper bound which is tighter.

We first derive a new upper bound for the regret ratio for a single CES utility function.

THEOREM 3. Let $f(p_i) = (\sum_{j=1}^d \alpha_j \cdot p_i \cdot c_j^b)^{\frac{1}{b}}$ be a CES utility function, where $0 < b < 1$ and $\alpha_j \geq 0$. The regret ratio $r_regret_{\mathcal{D}}(\mathcal{S}, f)$ of a set \mathcal{S} returned by MinVar satisfies

$$r_ratio_{\mathcal{D}}(\mathcal{S}, f) \leq \frac{(d-1)^{\frac{1}{b}}}{t + (d-1)^{\frac{1}{b}}}$$

PROOF. Let p^* be the point in \mathcal{D} with the largest utility computed by f , and s^* be the point in \mathcal{S} that is selected in the same bucket where p^* lies in. We have:

$$\begin{aligned} regret_{\mathcal{D}}(\mathcal{S}, f) &= \max_{p_i \in \mathcal{D}} f(p_i) - \max_{p_i \in \mathcal{S}} f(p_i) \\ &\leq f(p^*) - f(s^*) \\ &= (\sum_{j=1}^d \alpha_j \cdot p^* \cdot c_j^b)^{\frac{1}{b}} - (\sum_{j=1}^d \alpha_j \cdot s^* \cdot c_j^b)^{\frac{1}{b}} \end{aligned}$$

Since $g(x) = x^{\frac{1}{b}}$ is convex when $0 < b < 1$, we have $g(x) - g(y) \leq (x - y)g'(x)$. Thus,

$$\begin{aligned} regret_{\mathcal{D}}(\mathcal{S}, f) &\leq (\sum_{j=1}^d \alpha_j \cdot p^* \cdot c_j^b - \sum_{j=1}^d \alpha_j \cdot s^* \cdot c_j^b) \cdot \frac{1}{b} \cdot (\sum_{j=1}^d \alpha_j \cdot p^* \cdot c_j^b)^{\frac{1}{b}-1} \\ &= \frac{1}{b} \left[\sum_{j=1}^d \alpha_j (p^* \cdot c_j^b - s^* \cdot c_j^b) \right] (\sum_{j=1}^d \alpha_j \cdot p^* \cdot c_j^b)^{\frac{1}{b}-1} \\ &\leq \frac{1}{b} \left[\sum_{j=1}^{d-1} \alpha_j (p^* \cdot c_j^b - s^* \cdot c_j^b) \right] (\sum_{j=1}^{d-1} \alpha_j \cdot p^* \cdot c_j^b)^{\frac{1}{b}-1} \end{aligned}$$

Consider another function $g(x) = x^b$, which is concave when $0 < b < 1$, and $g'(x)$ is monotonically decreasing. According to Lagrange's Mean Value Theorem, there must exist some ξ between two values x and y , such that $x^b - y^b = (x - y) \cdot b \cdot \xi^{b-1}$. Thus, we have

$$\begin{aligned} p^* \cdot c_j^b - s^* \cdot c_j^b &\leq |p^* \cdot c_j - s^* \cdot c_j| \cdot b \cdot (\min\{p^* \cdot c_j, s^* \cdot c_j\})^{b-1} \\ &\leq \frac{c_j^\tau}{t} \cdot b \cdot c_j^{\tau b-1} = \frac{b}{t} c_j^{\tau b} \end{aligned}$$

Thus,

$$\begin{aligned} regret_{\mathcal{D}}(\mathcal{S}, f) &\leq \frac{1}{b} (\sum_{j=1}^{d-1} \alpha_j \frac{b}{t} c_j^{\tau b}) (\sum_{j=1}^{d-1} \alpha_j \cdot p^* \cdot c_j^b)^{\frac{1}{b}-1} \\ &\leq \frac{(d-1)}{t} \cdot \max_{j \in [1..d-1]} \{\alpha_j c_j^{\tau b}\} \cdot \left[(d-1) \cdot \max_{j \in [1..d-1]} \{\alpha_j c_j^{\tau b}\} \right]^{\frac{1}{b}-1} \\ &= \frac{(d-1)^{\frac{1}{b}}}{t} (\max_{j \in [1..d-1]} \{\alpha_j c_j^{\tau b}\})^{\frac{1}{b}} \\ &\leq \frac{(d-1)^{\frac{1}{b}}}{t} (\max_{j \in [1..d-1]} \{\sum_{l=1}^d \alpha_l \cdot p_j^* \cdot c_l^b\})^{\frac{1}{b}} \\ &= \frac{(d-1)^{\frac{1}{b}}}{t} \max_{j \in [1..d-1]} \{(\sum_{l=1}^d \alpha_l \cdot p_j^* \cdot c_l^b)^{\frac{1}{b}}\} \end{aligned}$$

Let $\sigma = \frac{(d-1)^{\frac{1}{b}}}{t}$. Then,

$$regret_{\mathcal{D}}(\mathcal{S}, f) \leq \sigma \max_{j \in [1..d-1]} \{(\sum_{l=1}^d \alpha_l \cdot p_j^* \cdot c_l^b)^{\frac{1}{b}}\}$$

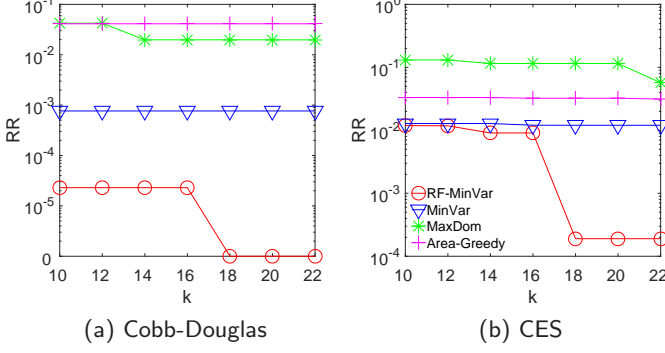


Figure 4: Varying k (NBA)

Since p_j^* ($j \in [1..d-1]$) is in \mathcal{S} , we have

$$\begin{aligned} \max_{p_i \in \mathcal{S}} f(p_i) &\geq \max_{j \in [1..d-1]} f(p_j^*) = \max_{j \in [1..d-1]} \left\{ \left(\sum_{l=1}^d \alpha_l \cdot p_j^* \cdot c_l^b \right)^{\frac{1}{b}} \right\} \\ &= \frac{1}{\sigma} \text{regret}_{\mathcal{D}}(\mathcal{S}, f). \end{aligned}$$

The regret ratio $r\text{-regret}_{\mathcal{D}}(\mathcal{S}, f)$ is hence bounded by

$$\begin{aligned} r\text{-regret}_{\mathcal{D}}(\mathcal{S}, f) &= \frac{\text{regret}_{\mathcal{D}}(\mathcal{S}, f)}{\text{regret}_{\mathcal{D}}(\mathcal{S}, f) + \max_{p_i \in \mathcal{S}} f(p_i)} \\ &= \frac{1}{1 + \frac{\max_{p_i \in \mathcal{S}} f(p_i)}{\text{regret}_{\mathcal{D}}(\mathcal{S}, f)}} \\ &\leq \frac{1}{1 + \frac{1}{\sigma}} = \frac{\sigma}{1 + \sigma} = \frac{(d-1)^{\frac{1}{b}}}{t + (d-1)^{\frac{1}{b}}} \end{aligned}$$

□

Therefore, given a set \mathcal{F} of CES functions, the maximum regret ratio $mr\text{-regret}_{\mathcal{D}}(\mathcal{S}, \mathcal{F})$ satisfies:

$$mr\text{-regret}_{\mathcal{D}}(\mathcal{S}, \mathcal{F}) \leq \frac{(d-1)^{\frac{1}{b}}}{t + (d-1)^{\frac{1}{b}}}$$

We can see from this bound that, when k decreases or d increases, the maximum regret ratio is expected to increase. For simplicity, we say that this bound grows in a scale of $O(\frac{1}{k^{\frac{1}{d-1}}})$. This bound is tighter than the bound $O(\frac{1}{bk^{\frac{1}{d-1}}})$ obtained in [10] since $0 < b < 1$.

To give an example, consider $\mathcal{F} = \{f_5, f_6\}$ where $f_5(p_i) = (0.5p_i \cdot c_1^{0.5} + 0.5p_i \cdot c_2^{0.5})^2$ and $f_6(p_i) = (0.99p_i \cdot c_1^{0.5} + 0.01p_i \cdot c_2^{0.5})^2$. Then $d = 2$. Let $k = 3$, which means $t = 2$. The upper bound of the maximum regret ratio $\frac{(d-1)^{\frac{1}{b}}}{t + (d-1)^{\frac{1}{b}}} = \frac{1}{2+1} \approx 33.33\%$. As k increases (e.g., to 20), this upper bound will decrease (e.g., to $\frac{1}{19+1} = 5\%$).

6. EXPERIMENTS

In this section we evaluate the empirical performance of the two proposed algorithms MinVar and RF-MinVar.

6.1 Settings

All the algorithms are implemented in C++, and the experiments are conducted on a computer running the OS X 10.11 operating system with a 64-bit 2.7 GHz Intel® Core™ i5 CPU and 8 GB RAM.

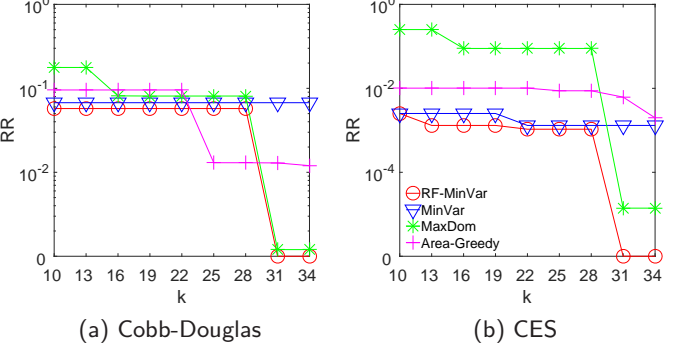


Figure 5: Varying k (Stocks)

Table 3: Experimental Settings

Parameter	Values	Default
Utility function	Cobb-Douglas, CES	-
Data set	NBA, Stocks, Anti-correlated	-
n	100, 1000, 10000, 100000, 1000000	10000
d	2..10	3
k	10..34	20

Both real and synthetic data sets are used in the experiments. The real data sets used are the *NBA*² and the *Stocks*³ data sets, which have been used in previous studies on k -regret queries [17, 18]. After filtering out data points with null fields, we obtain 20,640 data points of 7 dimensions in the NBA data set. The Stocks data set contains 122,574 data points of 5 dimensions.

The synthetic data sets are generated by the *anti-correlated* data set generator [1], which is a popular data generator used in skyline query studies [15, 16, 21]. We generate synthetic data sets with cardinality ranging from 100 to 1,000,000 and dimensionality ranging from 2 to 10. We vary the query parameter k from 10 to 34 in the experiments. Table 3 summarizes the parameters and the values used.

In each set of experiments, we randomly generate 10,000 different sets of parameters $\{\alpha_i | i \in [1..d], \alpha_i \in [0, 1]\}$ for each of the generalized Cobb-Douglas function and the CES function, where $\sum_{i=1}^d \alpha_i = 1$. The CES function has an extra parameter b . We generate random values of b in the range of $[0.1, 0.9]$. We run the algorithms on the data sets, and report the maximum regret ratio on the generated utility functions, denoted as *RR* in the result figures.

Five algorithms are tested in the experiments:

- *MinVar* is the algorithm proposed in Section 4.1. We use $inc = 0.01\%n$ to control the number of iterations that the sub-algorithm FindBreakpoints runs for.
- *RF-MinVar* is the improved MinVar algorithm powered by the heuristic proposed in Section 4.4. We find that $itr_{max} = 11$ is sufficient to handle the data sets tested. The results obtained are based on this setting.
- *MinWidth* is an algorithm proposed by Kessler Faulkner et al. [10] with bounded maximum regret ratios for k -regret queries with CES functions.

²<http://www.databasebasketball.com>

³<http://pages.swcp.com/stocks>

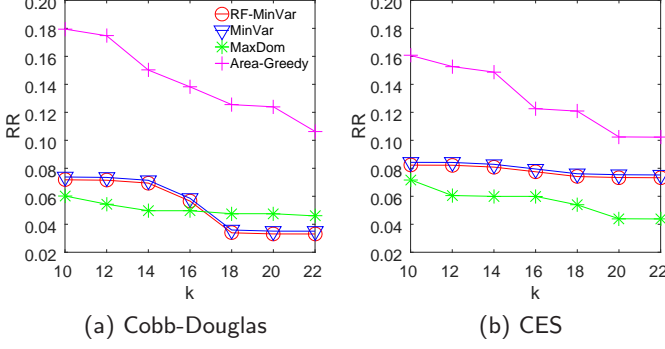


Figure 6: Varying k (Anti-correlated)

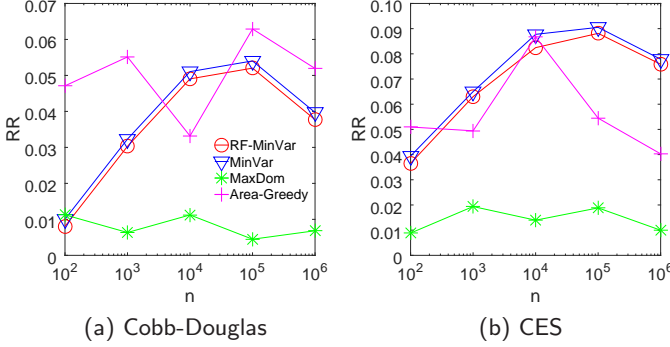


Figure 8: Varying n (Anti-correlated)

- *Area-Greedy* is a greedy algorithm proposed by Kessler Faulkner et al. [10] with good practical maximum regret ratios (but no bounds) for CES functions. Note that we do not compare with *Angle* [10] which is another greedy algorithm because it has been shown to produce larger regret ratios than those of *Area-Greedy*.
- *MaxDom* is a greedy algorithm proposed by Lin et al. [12] that returns the k representative skyline points which dominate the largest number of other points.

6.2 Results

Effect of k . We first test the effect of k on the maximum regret ratio. Figures 4 and 5 show the result where k is varied from 10 to 34 on the two real data sets. Note that Figure 4 shows up to $k = 22$ because the maximum regret ratios of the algorithms have become stable beyond this point. We also omit MinWidth as it is known to produce larger maximum regret ratios than those of Area-Greedy [10].

We can see from these two figures that, as k increases, the maximum regret ratios of the algorithms either decrease or stay stable. This confirms the upper bound derived. It is expected as a larger k means more points are returned and a higher probability of satisfying more utility functions. On the NBA data set, the proposed algorithm MinVar outperforms the baseline algorithms by more than an order of magnitude on the Cobb-Douglas functions, and more than 5 times on the CES functions (note the logarithmic scale). This is because the algorithm selects the k points from a much more evenly divided data space, and the points selected are much closer to the optimal points for the various utility functions. Powered by the redundant point removing heuristic, RF-MinVar achieves even smaller maximum

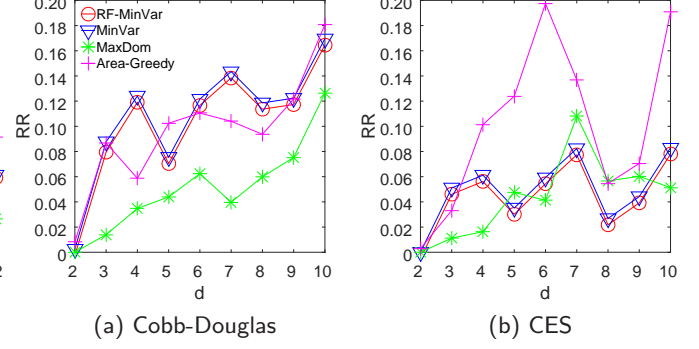


Figure 7: Varying d (Anti-correlated)

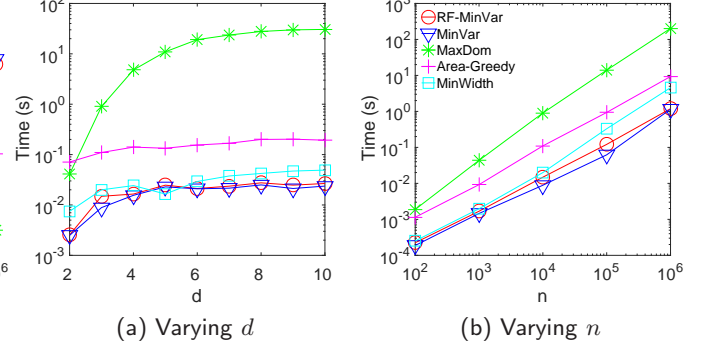


Figure 9: Running time tests (Anti-correlated)

regret ratios, as it avoids selecting the dominated points which do not contribute to the regret ratios. On the Stocks data set, similar patterns are observed. MinVar outperforms the baseline algorithms in most cases, and RF-MinVar outperforms the baseline algorithms in all cases tested. These confirm the superiority of the proposed algorithms.

In Figure 6 we show the result of varying k on a synthetic data set. On this data set, MaxDom obtains the smallest maximum regret ratios. This is because the data points in this synthetic data set follow an anti-correlated distribution. The data points tend *not* to be dominated by each other and tend to be optimal for different utility functions. It is intrinsically difficult to find a small number of data points that are optimal for a large number of utility functions under such data distribution. MaxDom is designed to handle this type of data while our algorithms are not. Note that, even under such an extreme case, the maximum regret ratios obtained by the proposed algorithms are still very close to those obtained by MaxDom, i.e., the difference is less than 0.04 (cf. Figure 6 (b)). Also, *MaxDom does not have a bound on the maximum regret ratio obtained while our algorithms do*. In the meantime, the proposed algorithms both again outperform the other baseline algorithm Area-Greedy.

Effect of d . Next we test the algorithm performance by varying the number of dimensions d from 2 to 10 with synthetic data. Figure 7 shows the result. Similar to Figure 6, on the synthetic data, MaxDom obtains the smallest maximum regret ratios, while those of the proposed algorithms are very close. Another observation is that, as d increases, the maximum regret ratios increase overall for all the algorithms. This again confirms the bounds obtained, and is expected as the difference between the optimal points in \mathcal{D} and \mathcal{S} accumulates as there are more dimensions. Fluctu-

ations are observed in the maximum regret ratios. This is because the maximum regret ratios are quite small already (i.e., less than 0.20). A change in the data set may have a random impact on the maximum regret ratios, even though the overall trend is increasing.

Effect of n . We further test the scalability of the proposed algorithms by varying the data set cardinality from 100 to 1,000,000. The comparative performance of the algorithms is shown in Figure 8, which again is similar to that shown in Figure 7. Note that, while the bounds of the *maximum* regret ratio do not change when n changes, the *actual* regret ratios obtained by the algorithms may change as n changes. This is natural as the optimal points have a smaller probability to be returned when there are more points in the data set. However, this change in the actual regret ratios is still bounded. We observe that, for the synthetic data sets with up to 1,000,000 data points, the maximum regret ratios of the proposed algorithms are below 0.10. This confirms the scalability of the proposed algorithms.

Running time results. We also show the running time of the algorithms on the synthetic data sets in Figure 9. Note that the utility functions are irrelevant in this set of experiments as the algorithms do not rely on any particular utility function. We see that the proposed algorithms are the fastest in almost all the cases tested. MinWidth is the only baseline algorithm with a relatively close performance, but it has much larger maximum regret ratios. The proposed algorithms outperform the other two baseline algorithms by more than an order of magnitude. MaxDom is the slowest. It takes almost 500 seconds to process 1,000,000 data points (cf. Figure 9 (b)), while the proposed algorithms can finish in just over a second. This again confirms the scalability of the proposed algorithms.

7. CONCLUSIONS

We overcame the barrier of multiplicative utility functions and presented an algorithm named MinVar to process k -regret queries with such functions. We showed that MinVar can produce query answers with a bounded maximum regret ratio. In particular, when applied on k -regret queries with Cobb-Douglas functions, MinVar achieved a maximum regret ratio bounded between $\Omega(\frac{1}{k^2})$ and $O(\ln(1 + \frac{1}{k^{\frac{1}{d-1}}}))$; when applied on k -regret queries with Constant Elasticity of Substitution functions, MinVar achieved a maximum regret ratio bounded between $\Omega(\frac{1}{bk^2})$ and $O(\frac{1}{k^{\frac{1}{d-1}}})$. We further proposed a heuristic to lower the maximum regret ratio obtained by MinVar, resulting in an improved algorithm named RF-MinVar. We performed extensive experiments using both real and synthetic data to evaluate the performance of both algorithms. The results showed that the regret ratios of the answers produced by MinVar and RF-MinVar are constantly small. Meanwhile, both algorithms are more efficient than the baseline algorithms.

This study opens up opportunities to explore k -regret queries with various other types of multiplicative utility functions. It would also be interesting to see how k -regret queries with a mix of both additive and multiplicative utility functions can be answered with a bounded regret ratio.

Acknowledgments

We thank Dr. Ashwin Lall for sharing the code of MinWidth and Area-Greedy, and providing feedback on our work.

8. REFERENCES

- [1] S. Börzsöny, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.
- [2] D. Brazianus and C. Boutilier. Minimax regret based elicitation of generalized additive utilities. In *The 23rd Conference on Uncertainty in Artificial Intelligence*, pages 25–32, 2007.
- [3] C.-Y. Chan, H. Jagadish, K.-L. Tan, A. K. Tung, and Z. Zhang. On high dimensional skylines. In *EDBT*, pages 478–495, 2006.
- [4] P. H. D. Charles W. Cobb. A theory of production. *The American Economic Review*, 18(1):139–165, 1928.
- [5] S. Chester, A. Thomo, S. Venkatesh, and S. Whitesides. Computing k -regret minimizing sets. *PVLDB*, 7(5):389–400, 2014.
- [6] P. Diamond, L. Helms, and J. Mirrlees. Optimal taxation in a stochastic economy. *Journal of Public Economics*, 14(1):1 – 29, 1980.
- [7] J. Huang, Z. Wen, J. Qi, R. Zhang, J. Chen, and Z. He. Top- k most influential locations selection. In *CIKM*, pages 2377–2380, 2011.
- [8] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top- k query processing techniques in relational database systems. *ACM Computing Surveys*, 40(4):11:1–11:58, 2008.
- [9] R. L. Keeney and H. Raiffa. *Decisions with multiple objectives: preferences and value trade-offs*. Cambridge university press, 1993.
- [10] T. Kessler Faulkner, W. Brackenbury, and A. Lall. k -regret queries with nonlinear utilities. *PVLDB*, 8(13):2098–2109, 2015.
- [11] K. Ligett and G. Piliouras. Beating the best Nash without regret. *SIGecom Exchanges*, 10(1):23–26, 2011.
- [12] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The k most representative skyline operator. In *ICDE*, pages 86–95, 2007.
- [13] D. Nanongkai, A. Lall, A. D. Sarma, and K. Makino. Interactive regret minimization. In *SIGMOD*, pages 109–120, 2012.
- [14] D. Nanongkai, A. D. Sarma, A. Lall, R. J. Lipton, and J. Xu. Regret-minimizing representative databases. *PVLDB*, 3(1-2):1114–1124, 2010.
- [15] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD*, pages 467–478, 2003.
- [16] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *VLDB*, pages 15–26, 2007.
- [17] P. Peng and R. C.-W. Wong. Geometry approach for k -regret query. In *ICDE*, pages 772–783, 2014.
- [18] P. Peng and R. C.-W. Wong. k -hit query: Top- k query with probabilistic utility function. In *SIGMOD*, pages 577–592, 2015.
- [19] M. A. Soliman, I. F. Ilyas, and K. C. C. Chang. Top- k query processing in uncertain databases. In *ICDE*, pages 896–905, 2007.
- [20] K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *VLDB*, pages 301–310, 2001.
- [21] Y. Tao, L. Ding, X. Lin, and J. Pei. Distance-based representative skyline. In *ICDE*, pages 892–903, 2009.
- [22] H. R. Varian. *Microeconomic analysis*. Norton & Company, 1992.
- [23] A. D. Vilcu and G. E. Vilcu. On some geometric properties of the generalized CES production functions. *Applied Mathematics and Computation*, 218(1):124–129, 2011.
- [24] G. E. Vilcu. A geometric perspective on the generalized Cobb-Douglas production functions. *Applied Mathematics Letters*, 24(5):777–783, 2011.
- [25] T. Xia, D. Zhang, and Y. Tao. On skylining with flexible dominance relation. In *ICDE*, pages 1397–1399, 2008.
- [26] S. Zeighami and R. C.-W. Wong. Minimizing average regret ratio in database. In *SIGMOD*, pages 2265–2266, 2016.